Guideline for Selecting and Tailoring a Life Cycle

Number: 580-GL-069-01
Effective Date: September 15, 2007
Expiration Date: September 15, 2012

Approved By: (signature)
Name: John Donohue
Title: Assoc. Division Chief

Responsible Office: Information Systems Division
Title: Guidelines for Selecting & Tailoring a Life Cycle

Asset Type: Guideline
PAL Number: 1.2.2.1

Purpose

The purpose of this document is to provide uniform guidance for the selection and possible tailoring of a software development life cycle.

Scope

This guideline should be used when defining the life cycle for a software development or maintenance effort.

Guideline

The software life cycle is the set of phases needed to complete the software development or maintenance process. The Life Cycle description identifies the order in which those phases are executed, the products expected from each phase, and the exit criteria indicating that the phase has been successfully completed.

Step 1: Identify the life cycle model to be used. There are several life cycle models that can be used as the basis for defining a software development life cycle, including the waterfall model, the incremental model, and the package-based model, among others. Each model has its strengths and weaknesses, and no one model is best for all situations. Each model has steps for requirements, design, implementation, testing, release to operations, maintenance, and retirement.

Five life-cycle models are summarized in Addendum A. These models are recommended on the basis of NASA's experience with applying them successfully at various centers. Development is addressed before maintenance, and the development life-cycle models are ordered from the simplest and most familiar to what may be the most complex and least familiar.

- Waterfall development life-cycle model
- Incremental development life-cycle model
- Evolutionary development life-cycle model
- Package-based development life-cycle model
- Legacy system maintenance life-cycle model

Another life cycle model may be used if deemed appropriate.

Step 2: Once a model is selected, the phases appropriate to the development effort must be identified. Determine what phases (if any) have already been completed. For example, if the requirements have been completed and a requirements review held, you may be able to begin at the

architecture or design phase.

Step 3: Once you have determined which phases are appropriate for your effort, tailor the definition of those phases as necessary. For each phase, determine the following:

- The environment that you establish for the phase (this will include the type of hardware or tools that will be necessary (e.g., hardware and tools for requirements definition, design, development, or test)).
- The list of products that you expect to produce as part of the phase (e.g., documents, data, and/or software)
- The list of stakeholders that you must involve during the phase (e.g., stakeholders from whom you need input, approval, or support).
- The exit criteria that signify the end of the phase (e.g., the appropriate end-of-phase review).

Document your tailored life cycle in your Product Plan. See Addendum B for an example life cycle description. Remember to check that the work processes (such as design or test processes) required for each life cycle phase of the effort have been identified (and tailored where necessary.)

Measures

Recommended Measures: No measures have been identified for selecting and tailoring a life cycle activity. However, measures should be defined for activities within each life cycle phase as appropriate. Those measures are identified in the Measurement and Analysis section of the Software Management Plan/Product Plan (SMP/PP).

Required Measures: None

References

For further information on this topic, please see:

- NASA Software Management Guidebook, NASA-GB-001-96, November 1996
- Standard for Developing a Software Project Life Cycle Process, IEEE P1074/D 5.5, January 2006

Tools and Templates

The following Tools are available to assist in defining the life cycle phases.

Name	Description
Software Management Plan/Product Plan Boilerplate Tool	The SMP/PP boilerplate contains a life cycle description that can be easily tailored for your life cycle. It documents the Incremental Life Cycle Model by default.

Change History

Version	Date	Description of Improvements
1.0	9/13/07	Initial version approved by CCB

Addendum A
Summary of Recommended Life Cycle Models

1. Waterfall Development Life-Cycle Model

The waterfall (single-build) life-cycle model is essentially a once-through, do-each-step-once approach. Simplistically, determine user needs, define requirements, design the system, implement the system, test, fix, and deliver the system.

Advantages:

- Well-studied, well-understood, and well-defined
- Easy to model and understand
- Easy to plan and monitor
- Many management tools exist to support this life-cycle model

Disadvantages

- Most if not all requirements must be known up front
- Does not readily accommodate requirements changes
- Product is not available for initial use until the project is nearly done

Most appropriate when ...

- Project is similar to one done successfully before
- Requirements are quite stable and well-understood
- The design and technology are proven and mature
- Total project duration is relatively short (less than a year)
- Customer does not need any interim releases

Major products and milestone reviews for this life-cycle model are summarized below.

Life-cycle phase	Major products	Milestone reviews
Project planning	▪ Software plan	None
Requirements definition and analysis	▪ Software requirements document	Software Requirements Review (SRR)
Architectural design	▪ Preliminary software design document ▪ Test plan ▪ Preliminary user's guide	Preliminary Design Review (PDR)
Detailed design	▪ Detailed software design document ▪ Requirements Traceability Matrix	Critical Design Review (CDR)
Implementation and testing	▪ Unit-level design ▪ Implemented, tested software ▪ Acceptance test procedures ▪ Draft user's guide ▪ Updated Requirements Traceability Matrix	Test Readiness Review (TRR)
Acceptance testing	▪ Acceptance -tested software ▪ Acceptance test report ▪ Final user's guide ▪ As-built software description	Acceptance Test Readiness Review (ATRR)

2. Incremental Development Life-Cycle Model

The incremental (multi-build) life-cycle model determines user needs and defines a subset of the system requirements, then performs the rest of the development in a sequence of builds. The first build incorporates part of the planned capabilities, the next build adds more capabilities, and so on, until the system is complete. This has been called a build-a-little, test-a-little approach.

Advantages

- Reduces risks of schedule slips, requirements changes, and acceptance problems
- Increases manageability
- Interim builds of the product facilitate feeding back changes in subsequent builds
- Interim builds may be delivered before the final version is done; this allows end users to identify needed changes
- Breaks up development for long lead time projects
- Allows users to validate the product as it is developed
- Allows software team to defer development of less well understood requirements to later releases after issues have been resolved
- Allows for early operational training on interim versions of the product
- Allows for validation of operational procedures early
- Includes well-defined checkpoints with customer and users via reviews

Disadvantages

- Like the waterfall life-cycle model, most if not all requirements must be known up front
- Sensitive to how specific builds are selected
- Places products (particularly requirements) under configuration control early in the life cycle, thereby requiring formal change control procedures that may increase overhead, particularly if requirements are unstable

Most appropriate when ...

- Project is similar to one done successfully before
- Most of the requirements are stable and well-understood; but some TBDs may exist
- The design and technology are proven and mature
- Total project duration is greater than one year or customer needs interim release(s)

Major products and milestone reviews for this life-cycle model are summarized below.

Life-cycle phase	Major products	Milestone reviews
Concept Definition	▪ Software plan ▪ Operations concept	None
Requirements Definition	▪ Software requirements document ▪ Requirements traceability matrix (RTM)	Software Requirements Review (SRR)
Preliminary Design	▪ System software design ▪ Test plan ▪ Preliminary user's guide ▪ Updated Requirements Traceability Matrix	Preliminary Design Review (PDR)
Detailed Design	▪ Detailed software design document (through at least the first build) ▪ Updated Requirements Traceability Matrix ▪ Software design document	Critical Design Review (CDR)

	update (detailed through at least the next build)) ▪ Updated Requirements Traceability Matrix	
Implementation	▪ Unit test drivers, unit test plans, unit test data, and unit test reports ▪ Coded, unit tested, integrated, and integration tested software executables Build integration test procedures and test results ▪ Draft user's guide ▪ Updated Requirements Traceability Matrix ▪ Final Test Plan ▪ System Description Document (Draft)	None
Build Testing	▪ Build test procedures, test results, and test report ▪ Problem reports	Test Readiness Review (TRR)
System Testing	▪ Build-tested software ▪ System test procedures, test results, and test report ▪ Problem reports ▪ Final user's guide ▪ As-built software description ▪ Delivery Package	Acceptance Test Readiness Review (ATRR)

3. Evolutionary Development Life-Cycle Model

Like the incremental development model, the evolutionary life-cycle model also develops a system in builds, but differs from the incremental model in acknowledging that the user needs are not fully understood and not all requirements can be defined up front. In the evolutionary approach, user needs and system requirements are partially defined up front, and then refined in each succeeding build. The system evolves as the understanding of user needs and the resolution of issues occurs. Prototyping is especially useful in this life-cycle model.

The evolutionary development life-cycle model is sometimes referred to as a spiral development model. This model is also sometimes referred to as a prototyping life-cycle model, but it should not be confused with the prototyping technique. The evolutionary development is not usually appropriate for development of critical mission software, since there is more difficulty in managing it and measuring progress when using it.

Advantages

- Not all requirements need be known up front
- Addressing high risk issues (for example, new technologies or unclear requirements) early may reduce risk
- Like the incremental life-cycle model, interim builds of the product facilitate feeding back changes in subsequent builds
- Users are actively involved in definition and evaluation of the system

- Prototyping techniques enable developers to demonstrate functionality to users with minimal effort
- Even if time or money runs out, some amount of operational capability is available

Disadvantages

- Because not all requirements are well-understood up front, the total effort involved in the project is difficult to estimate early. Therefore, expect accurate estimates only for the next cycle, not for the entire development effort.
- Less experience on how to manage (progress is difficult to measure)
- Risk of never-ending evolution (for example, continual "gold plating")
- May be difficult to manage when cost ceilings or fixed delivery dates are specified
- Will not be successful without user involvement

Most appropriate when ...

- Requirements or design are not well-defined, not well-understood, or likely to undergo significant changes
- New or unproved technologies are being introduced
- System capabilities can be demonstrated for evaluation by users
- There are diverse user groups with potentially conflicting needs

Major products and milestone reviews for this life-cycle model are summarized below.

Life-cycle phase	Major products	Milestone reviews
Concept Definition	▪ Initial System Development Plan to be updated in later phases	System Concept Review (SCR)
Requirements and architecture definition	▪ Preliminary requirements document ▪ Architectural design document containing the infrastructure plus the architecture of each release as it evolves ▪ Requirements Traceability Matrix	Combined System Requirements Review (SRR) and System Design Review (SDR)
Implementation	▪ Evolutionary Implementation Plan ▪ Iteration plan for each iteration ▪ Software product baseline combining new, reused, and off-the-shelf products ▪ Updated Requirements Traceability Matrix ▪ Draft user documentation	Iteration assessments Release testing after all interations for the release have been completed
Release Integration and Test	▪ Release test procedures ▪ Integrated, tested software) ▪ Updated Requirements Traceability Matrix ▪ Release test report ▪ Final user documentation	Acceptance Test Readiness Review (ATRR)
Installation and acceptance	These system life-cycle phases are identified for completeness but are out of the scope of the software development life cycle.	
Operations and maintenance		

4. Package-Based Development Life-Cycle Model

The package-based development life-cycle model is used for system development based largely on the use of commercial-off-the-shelf and Government off-the-shelf products and reusable packages. Typically, some custom software development is needed to provide interfaces among the non-developed items (NDIs).

Advantages

- Lower cost than developing equivalent functionality from scratch
- Cycle time also often lower than developing equivalent functionality from scratch
- Improves confidence in quality of the end product (since quality of NDIs is already known)

Disadvantages

- May result in compromises between desired functionality and functionality provided by NDIs
- Maintainability may be more of a challenge because source of NDIs may not be the same NASA organization (for example, requires third party to make changes, raises source configuration management issues when NDI vendor releases updated versions)

Most appropriate when ...

- A significant portion of the functionality of a system can be provided by NDIs

Major products and milestone reviews for this life-cycle model are summarized below.

Life-cycle phase	Major products	Milestone reviews
Requirements Analysis and Package Identification	▪ System Development Plan ▪ Requirements document ▪ Strawman high-level architecture ▪ Candidate packages	System Requirements Review (SRR)
Architectural Definition and Package Selection	▪ Modified requirements document ▪ System architecture ▪ Final packages	System Design Review (SDR)
System Integration and Test	▪ Delivered system User demonstrations	Operational Readiness Review (ORR)
Technology Update and System Maintenance	▪ Enhanced system	User demonstrations

5. Legacy System Maintenance Life-Cycle Model

The legacy system maintenance life-cycle model is used to apply fixes or minor enhancements to an operational system. (Use a waterfall or incremental life-cycle model for major enhancements.) Selected and sometimes abbreviated activities performed in the software *development* life cycles are also performed during maintenance. The legacy system maintenance life-cycle model is similar in nature to the waterfall life-cycle model; the primary difference is that the architectural design has already been established.

Most appropriate when ...

- Maintenance release comprises only fixes and minor enhancements.

Major products and milestone reviews for this life-cycle model are summarized below.

Life-cycle phase	Major products	Milestone reviews
Release planning	▪ Release contents agreement	Release Contents Review (RCR)
Requirements definition and analysis	▪ Release requirements document	Release Requirements Review (RRR)
Design	▪ Release design document	Release Design Review (RDR)
Implementation and testing	▪ Unit-level design ▪ Implemented, tested software ▪ Release test plan and procedures ▪ Draft user's guide updates	Release Test Readiness Review (TRR)
Release testing	▪ Release-tested software ▪ Release test report ▪ Final user's guide updates ▪ As-built software description updates	Acceptance Test Readiness Review (ATRR)

Addendum B
Example Software Development Life Cycle Description

After you have selected and tailored the life cycle model for your effort, document the life cycle by describing the model and each applicable life cycle phase. The following table is an example life cycle description for the Incremental Development Life Cycle Model. Use this example as a guide for the format and level of detail required. Create your table by using the table below as a template. Delete any phases that are not part of your life cycle model or are not applicable to your effort. For each phase insert the information on: the environment needed for the phase; the products that you expect to produce during the phase; the stakeholders that you must involve; and the exit criteria that signify the end of the phase.

Note that the software development life cycle is a subset of the overall system life cycle. The software development life cycle as shown in this table is consistent with the System Development Life Cycle defined in NPR 7123.1.

Phase	Development Activity
Concept Definition	The PDT formulates a conceptual architecture by working with the customer to understand the system concept and high-level requirements and by looking for similarities to previous systems. To the extent possible, the PDT uses his/her experience to guide the customer in making trades affecting (or affected by) the system. The PDT develops an initial re-use strategy by identifying existing software including commercial off the shelf (COTS) and Government off the shelf (GOTS) software for potential use. The PDT also develops a preliminary acquisition strategy by performing an initial make/buy analysis. The results of this analysis document what portions of the system, if any, will be considered for potential acquisition. Planning activities are performed, including the definition of the WBS, estimation of cost and effort, initial risk identification, planning for adequate skilled staff. Budget and schedule are negotiated with the customer. If portions of the system will potentially be acquired, the activities described in the acquisition life cycle are initiated.
Environment	• Requirements/Design Environment
Products	• Operations concept • Concept review materials • SMP/PP signed and approved by signatories
Major Stakeholders	• PDT • Customer Management • Branch Management • Users
Exit Criteria	• Concept review materials are reviewed and approved • Action Items are assigned and scheduled for completion

Phase	Development Activity
Requirements Definition	The PDT works closely with the customer to understand and document the software requirements for the system. Detailed software requirements, including interface and performance requirements, are derived from customer requirements and documented. The detailed requirements are analyzed and reviewed for completeness, consistency, feasibility, and testability. Requirements that potentially drive software design decisions (e.g., special timing requirements, checkpoint restart) are identified. A requirements traceability matrix (RTM) is produced, showing how the high-level requirements flow down to detailed requirements. The initial re-use strategy, risks, and planning parameters (e.g., schedules, staffing, budgets, build/release plan) are reviewed and updated as appropriate.
Environment	• Requirements/Design Environment
Products	• Requirements Document(s) • Requirements traceability matrix (RTM) • Software Requirements Review (SRR) Materials
Major Stakeholders	• PDT • Customer Management • Branch Management • Users
Exit Criteria	• SRR materials are reviewed and approved • SRR Request For Action (RFAs) are assigned and scheduled for completion • Requirements are baselined and placed under configuration management.
Design	Working from the requirements and conceptual design, requirements are allocated to major subsystems. All internal and external interfaces are defined to the subsystem level. Designs of high-level functions or objects are specified and peer reviewed. The PDT refines and extends the software architecture down to the unit level. By successive refinement, they elaborate the design to produce “code-to” specifications for each unit. Selection of components and units for re-use are finalized. The RTM is updated to show how the requirements flow down to design components and software builds.
Environment	• Requirements/Design Environment
Products	• System Design • Test Plan (Draft) • Preliminary Design Review (PDR) Materials • Critical Design Review (CDR) Materials
Major Stakeholders	• PDT • Customer Management • Branch Management • Users

Phase	Development Activity
Exit Criteria	• PDR materials are reviewed and approved • PDR RFAs are assigned and scheduled for completion • CDR materials are reviewed and approved • CDR RFAs are assigned and scheduled for completion
Implementation	New units are coded from design specifications and required modifications to re-used units are made according to standards identified as applicable to the effort. Unit test plans and procedures with expected results are written. New and modified units are unit tested according to applicable standards identified for the effort. Unit test reports are written. System components are integrated into a build. Build-integration test procedures are written, executed, and test results are reviewed. Problem reports are written and corrected as appropriate. "As-built" system documentation is drafted. The RTM is updated to show how the detailed design flows down to software units.
Environment	• Development Environment
Products	• Unit test drivers, unit test plans, unit test data, and unit test reports • Build integration test procedures and test results • Test Plan (Final) • User's Guide (Draft) • System Description Document (Draft) • Coded, unit tested, integrated, and integration tested software executables
Major Stakeholders	• PDT • Customer Management • Branch Management • Users
Exit Criteria (for each build)	• Unit tests of each unit for a specified build are successfully executed in accordance with the unit test plan. • Build integration tests are successfully executed in accordance with approved build-integration test procedures. • Build-integration test reports are reviewed and approved.
Build Test	Build test scenarios and test procedures are written, peer-reviewed, and corrected. Requisite test software and/or test equipment is configured, tested, and verified for adequacy and correctness. Build tests are conducted according to the test plan and associated procedures. Build test reports are written and reviewed. Problem reports are written and corrected as appropriate. The RTM is updated to show how the detailed design flows down to build tests.
Environment	• Test Environment

Phase	Development Activity
Products	• Build test procedures, test results, and test report • Problem reports
Major Stakeholders	• PDT • Customer Management • Branch Management • Users
Exit Criteria (for each Build)	• All build tests for the build are executed. • All Build test results are analyzed, approved, and archived • Problem reports are submitted • Build Test Report is completed
System Test	System test scenarios and detailed test procedures are written, peer-reviewed, and corrected. Requisite test software and /or test equipment is configured, tested, and verified for adequacy and correctness. System tests are conducted according to the Test Plan and associated procedures. System test results are recorded and a System Test Report is written and reviewed. Problem reports are written and corrected as appropriate. The RTM is updated to show how the detailed design flows down to system tests. The system and its associated documentation are packaged for delivery.
Environment	• Test Environment
Products	• System test procedures, test results, and test report • Problem reports • Acceptance Test Readiness Review (ATRR) materials • Delivery Package
Major Stakeholders	• PDT • Customer Management • Branch Management • Users
Exit Criteria	• Final system tests are successfully executed • Build Test Report is completed • ATRR materials are reviewed and approved • Problem reports are submitted • All critical problem reports are implemented, delivered, verified and validated.
Post-Delivery Test Support	The PDT provides technical support such as consulting and analyzing test results as required during post-delivery test support. In the event that any changes are required as a result of these tests, problem reports may be generated and approved. Following design and implementation of corrections for these problem reports, Build Test and System Test will be repeated, with test upgrades as necessary to re-validate the updated system.

Phase	Development Activity
Products Major Stakeholders	• Updated and tested software executables (as needed) • Documentation updates (as needed) • PDT • Customer Management • Branch Management • Users